Domain-Specific Architectures: Research Problems and Promising Approaches

ANISH KRISHNAKUMAR and UMIT OGRAS, University of Wisconsin–Madison
RADU MARCULESCU, The University of Texas at Austin
MIKE KISHINEVSKY, Intel Corporation
TREVOR MUDGE, University of Michigan

Process technology-driven performance and energy efficiency improvements have slowed down as we approach physical design limits. General-purpose manycore architectures attempt to circumvent this challenge, but they have a significant performance and energy-efficient gap compared to special-purpose solutions. Domain-specific architectures (DSAs), an instance of heterogeneous architectures, efficiently combine general-purpose cores and specialized hardware accelerators to boost energy efficiency and provide programming flexibility. Indeed, the hardware, software, and systems aspects in DSAs are highly tailored to maximize the energy efficiency of applications in a target domain. As DSAs and their conceptualization advance rapidly, there is a strong need to understand the research problems that need immediate attention. This article discusses the primary research directions in the design and runtime management of DSAs. Then, it surveys some promising approaches and highlights the outstanding research needs.

CCS Concepts: • **Computer systems organization** → **Architectures**; **System on a chip**; • **Hardware** → **Emerging architectures**; **On-chip resource management;**

Additional Key Words and Phrases: Domain-specific architectures, domain-specific system-on-chip, DSA runtime resource management, hardware architectures, emerging systems, runtime frameworks

ACM Reference format:
Anish Krishnakumar, Umit Ogras, Radu Marculescu, Mike Kishinevsky, and Trevor Mudge. 2023. Domain-Specific Architectures: Research Problems and Promising Approaches. *ACM Trans. Embedd. Comput. Syst.* 22, 2, Article 28 (January 2023), 26 pages.
https://doi.org/10.1145/3563946

1 INTRODUCTION

Process technology-driven power, performance, and energy efficiency improvements have recently slowed down significantly [74, 166]. In addition, performance cannot be further improved by scaling the frequency arbitrarily due to the power wall [55, 180]. Consequently, two primary

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7860. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.
Authors' addresses: A. Krishnakumar and U. Ogras, University of Wisconsin–Madison; emails: {anish.n.krishnakumar, uogras}@wisc.edu; R. Marculescu, The University of Texas at Austin; email: radum@utexas.edu; M. Kishinevsky, Intel Corporation; email: michael.kishinevsky@intel.com; T. Mudge, University of Michigan; email: tnm@umich.edu.

1539-9087/2023/01-ART28 $15.00
https://doi.org/10.1145/3563946

drivers of higher performance-per-watt cease to provide the expected gains. At the same time, the instruction-level parallelism techniques, such as processor pipelining, prefetching, and out-of-order execution, provide only marginal benefits, thereby leaving a substantial scope for improvement in performance and energy efficiency [95].

Homogeneous multicore architectures integrate multiple identical cores onto the same die to provide higher computational capabilities under similar area budgets [66, 93]. They opened new avenues to parallel processing capabilities with higher performance at a modest power consumption increase, thereby allowing drastic energy efficiency improvements [68]. However, homogeneous cores cannot simultaneously satisfy competing application requirements, such as low power and high performance. Low-power cores, such as the Arm Cortex-M series, have limited performance. In contrast, high-performance cores, such as the Arm Cortex-A72/A76 processors, consume higher power due to the out-of-order execution nature, large caches, and deep execution pipelines. Heterogeneous multiprocessor architectures address this problem by integrating low-power and high-performance cores [83, 135]. Therefore, heterogeneous architectures are extensively used in most processing systems, such as mobile phones, laptops, desktops, and servers [5, 82, 127].

Heterogeneous architectures significantly improve performance and energy efficiency compared to their homogeneous counterparts. However, they still have a substantial gap with **application-specific integrated circuits (ASIC)**. To provide a quantitative comparison, Figure 1(a) shows the energy efficiency of applications implemented on CPU, GPU, FPGA, and ASIC. CPU implementations require the least design effort and also provide low energy efficiency [187]. GPUs and FPGAs improve energy efficiency and performance by exploiting **single-instruction multiple data (SIMD)** execution and parallelism benefits, respectively [59, 151]. Application code is converted to GPU-compatible code to run on GPUs, and hardware description languages or high-level synthesis for FPGAs. ASICs provide the highest energy efficiency since they are specifically designed for the target application [85]. However, the ASIC effort, which includes design, development, fabrication, and software development, could require several months to years. Therefore, there is a critical need to continue the evolution of computing architectures to provide ASIC-like energy efficiency with the shortest possible time-to-market.

Domain-specific architectures (DSAs) represent an emerging instance of heterogeneous architectures that optimize data flow for applications in a target domain through hardware acceleration while providing programming flexibility [6, 10, 86]. Examples of recently growing domains include machine learning and **artificial intelligence (AI)**. For instance, machine learning and AI are extensively being used for image processing, scheduling, recommendation systems, spam filtering, stock market analysis, and medical applications [120, 134, 142, 183]. Hence, there is a strong need for computing architectures that enable seamless, high-performance, and energy-efficient execution of these domain applications. DSAs aim at improved programmability by including general-purpose cores and the highest energy efficiency by integrating special-purpose processors and hardware accelerators. The domain-specific nature of DSAs stems from the fact that the hardware accelerators and data flows are highly tailored to the type of computations in the applications of a particular domain. Broadly speaking, DSAs encompass any computing architecture that provides the following:

- *Superior energy efficiency through specialized processing:* The specialized processors accelerate the frequently occurring domain-specific computations in hardware, thereby boosting energy efficiency. For example, a custom-designed **fast Fourier transform (FFT)** hardware accelerates the direct- and inverse-FFT operations, whereas a systolic matrix multiplication processor accelerates machine learning and AI applications.

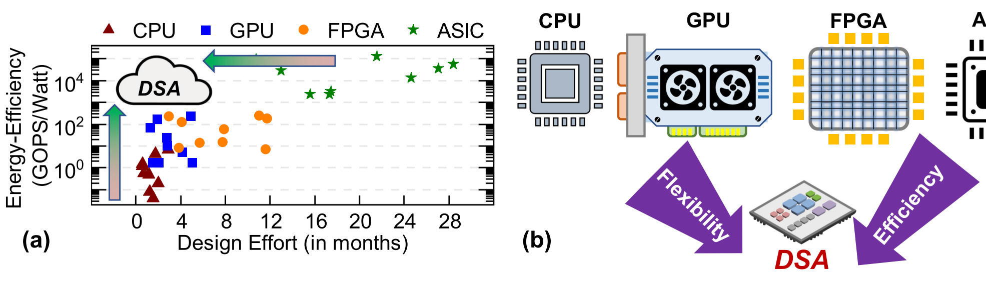

Fig. 1. (a) Trends in energy efficiency and design effort in giga operations per second per watt (GOPS/watt) for applications implemented on CPU, GPU, FPGA, and fixed-function/special-purpose ASIC. (b) An illustration of a domain-specific architecture (DSA) combining the flexibility benefits of CPU and GPU implementations, the performance benefits of FPGA, and the energy efficiency of fixed-function ASIC implementations.

- *Programmability/flexibility:* DSAs aim to improve the programming flexibility for both domain and non-domain applications. For example, DSAs that target neural network inference must be programmable to execute multilayer perceptrons, convolutional neural networks, and recurrent neural networks. In addition, they must be capable of executing other neural network inference operations that cannot be easily implemented using specialized hardware. Finally, they should be able to execute non-domain applications to improve flexibility and enable broader usage.
- *Heterogeneous processing elements:* The diverse types of **processing elements (PEs)** in DSAs cater to contrasting application requirements such as low power, high performance, energy efficiency, and programmability.

The potential of DSAs is also evident in recent and growing commercial examples. Google's tensor processing unit comprises hardware designs, systems, and software stacks to accelerate machine learning training and inference [133, 138]. Tensor processing units provide 3× to 7× speedup over state-of-the-art GPUs and 80× better energy efficiency than general-purpose processors [86, 96, 181]. Nvidia's **data center processing unit (DPU)** is another DSA that integrates high-performance ARM cores and hardware accelerators with an extensive software eco-system optimized for AI, cloud supercomputing, network security, and wireless communication [32]. Intel's **infrastructure processing unit (IPU)** is a programmable network device that integrates with server CPUs to accelerate networking control, storage management, and security. Offloading the infrastructure operations to the infrastructure processing unit reduces the overhead of infrastructure tasks to improve overall performance and energy consumption [31]. In the low-power domain, RedMulE offers a sub-100 mW DSA for deep learning that comprises RISC-V cores and dedicated matrix-multiplication accelerators [168]. In summary, DSAs have started making substantial strides in all domains to offer superior energy efficiency and short time-to-market. This article aims to discuss the primary research directions in DSAs and survey the academic work performed in these directions. Section 2 overviews the research directions in DSAs.

2 OVERVIEW OF PRIMARY RESEARCH DIRECTIONS IN DSAs

DSAs have the potential to enable high energy efficiency and programmability across multiple applications. However, critical research and infrastructure design challenges must be addressed before DSAs can become a mainstream computing paradigm [74, 86, 119]. For instance, designers must choose the optimal number and type of PEs to balance design time, cost, complexity, area, and energy efficiency. Novel and rapid hardware design techniques that condense the design

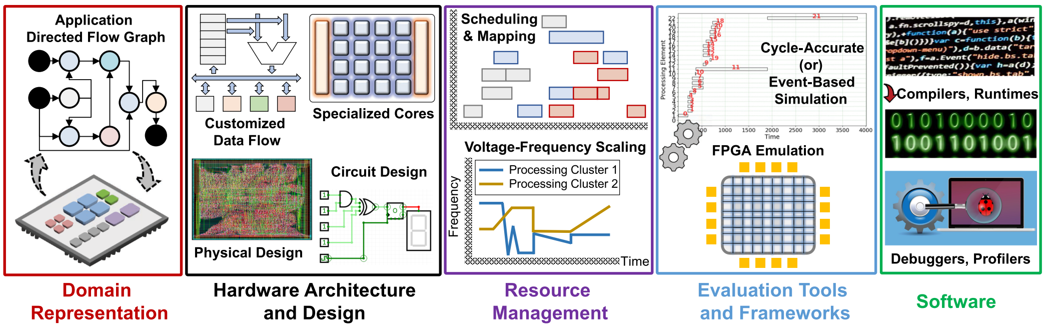

Fig. 2. Prime research directions in the conceptualization, design, and development of DSAs. Applications are represented as directed flow graphs. The nodes in the graph represent the key computational kernels within each application, and the edges of the graph denote the communication volumes between kernels.

time and costs allow for a shorter time-to-market [72, 119]. Similarly, DSAs require novel and state-of-the-art simulation, compilation, and emulation frameworks to minimize the gap between conceptualization and market availability of a product [170]. In summary, there is a strong need to understand the factors that currently limit the design and deployment of DSAs. To this end, this survey article identifies the key research areas (summarized in Figure 2) that need new ideas and solutions to make DSAs default choices for designers, developers, and end users:

- *Domain representation:* Application source code must be analyzed to extract the domain-specific kernels and construct the data flow graphs that can exploit the data- and task-level parallelism both in applications and hardware [171]. Understanding the domain applications plays a critical role in selecting the PEs for the DSA, as described in Section 3.
- *Hardware architecture and design:* With the saturation of energy efficiency of general-purpose processors, DSAs require novel hardware architectures and innovative solutions to exploit parallelism and maximize energy efficiency for domain-specific kernels. Section 4 discusses hardware architectures for DSAs.
- *Resource management in DSAs:* Exploiting the full potential of DSAs involves optimally allocating the tasks to PEs, and selecting their voltage-frequency levels at runtime using resource management algorithms described in Section 5.
- *Evaluation frameworks and productivity tools:* Section 7 presents the need and frameworks for rapid design space exploration to aid top-level design decisions in the early development phase and emulation platforms to aid functional validation and software development in DSAs.
- *Software development:* The challenge in programming DSAs with heterogeneous PEs demands innovation in software frameworks and toolchains. Section 6 also serves as a bridge between techniques for domain representation, hardware design, and resource management.

The rest of the article is organized as follows. Section 3 presents the ideas and directions of focus under domain representation. Section 4 discusses energy-efficient hardware architectures for DSAs. The resource management techniques and algorithms to exploit the potential of these designs are discussed in Section 5. Section 6 presents the software development aspects, whereas Section 7 surveys the tools and evaluation frameworks that aid the design process from the conceptualization phase to the end product. Section 8 discusses the interaction between the various DSA research directions. Section 9 concludes the survey.

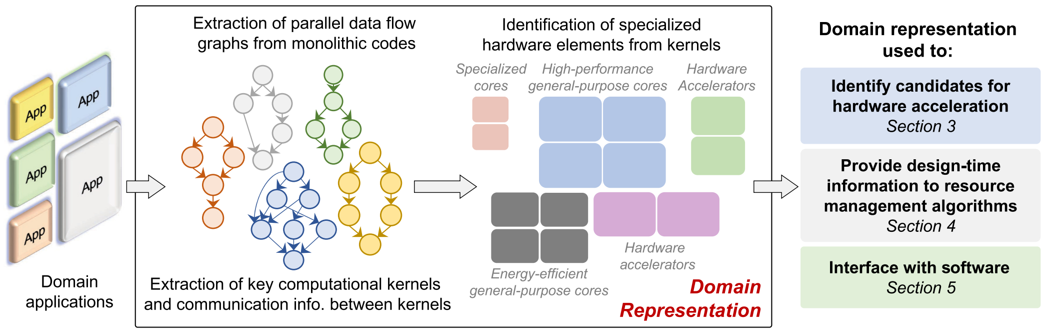
App
Domain applications
Extraction of parallel data flow graphs from monolithic codes
Extraction of key computational kernels and communication info. between kernels
Identification of specialized hardware elements from kernels
Specialized cores
High-performance general-purpose cores
Hardware Accelerators
Energy-efficient general-purpose cores
Hardware accelerators
Domain Representation
Domain representation used to:
Identify candidates for hardware acceleration Section 3
Provide design-time information to resource management algorithms Section 4
Interface with software Section 5

Fig. 3. Domain representation techniques extract the key application indicators (e.g., kernels and data flow graphs) and facilitate the design of hardware, software, and resource management techniques. The nodes in the data flow graph represent the kernels/tasks in the applications. The edges represent the dependencies between the kernels, and the edge weights denote the amount of data transferred between the kernels.

3 DOMAIN REPRESENTATION

The design of DSAs critically depends on *first* analyzing applications to classify them into the target domains. Then, structured information (also called *computational kernels*) is extracted from these applications of interest (domain applications) [13, 74, 123]. The frequently occurring computation kernels in an application domain are potential candidates for specialized implementations since benefits over repeated operations can lead to significant overall savings. Extracting the flow graph of an application provides precise control and data flow dependencies, thereby allowing the computing platforms to exploit the inherent parallelism and maximize performance [57]. The nodes of the flow graph represent the computational kernels, the edges between the nodes represent the dependency between the different kernels, and the weight of the edges denotes the volume of data communicated between two kernels, as shown in Figure 3 [27, 170]. The tasks that can be performed together present opportunities for parallel execution in DSAs. The information about the critical kernels and the scoped applications is utilized to determine the number and type of PEs [46]. The domain representation techniques for DSAs draw their inspiration from parallel computation models and languages [156]. The study of these techniques constitutes the domain representation research in DSAs, as overviewed in Figure 3.

Applications and domains are evolving at an unprecedented pace, making manual analysis of the applications arduous and impractical. Examples include the use of computer vision and image processing algorithms in autonomous driving applications, wireless and radar applications in communication and surveillance, and machine learning in AI [12, 27, 28, 105]. Hence, there is a strong need for frameworks that can automatically scope the application domain and extract the kernels and the data-parallel flow graphs [27]. The kernel and flow graph information help designers determine the number and type of PEs required by the DSA. Application tracing tools allow developers to log data as the programs execute, which is utilized to extract the kernel and flow graph information [36]. For instance, the **low-level virtual machine (LLVM)** compiler backend uses application instrumentation to construct an intermediate platform-independent representation and generate application traces [188]. The kernel information in the applications and the data flow graphs are extracted from the traces [27, 28, 100, 170]. This flow enables chip developers to make intelligent choices of the hardware elements that can maximize energy efficiency.

Heterogeneous **systems-on-chip (SoCs)**, especially hardware accelerator rich systems, face an enormous challenge in moving data among the different PEs [73]. **Networks-on-chip (NoCs)** are typically employed to improve the on-chip communication latencies [118]. Critical

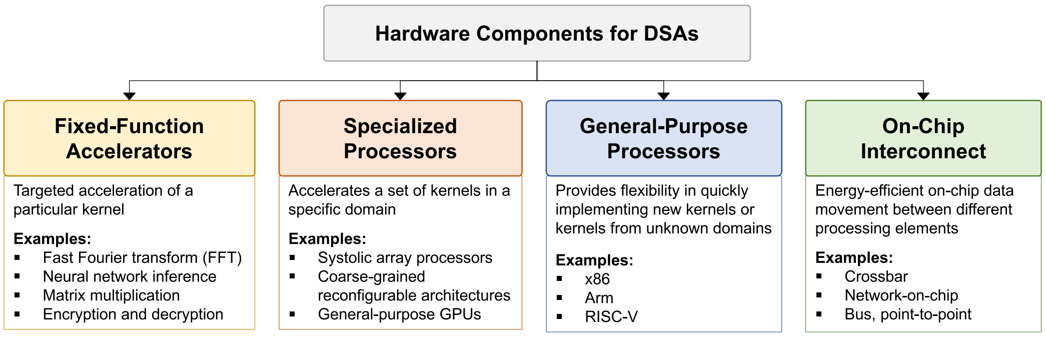
Hardware Components for DSAs
Fixed-Function Accelerators
Targeted acceleration of a particular kernel
Examples:
Fast Fourier transform (FFT)
Neural network inference
Matrix multiplication
Encryption and decryption
Specialized Processors
Accelerates a set of kernels in a specific domain
Examples:
Systolic array processors
Coarse-grained reconfigurable architectures
General-purpose GPUs
General-Purpose Processors
Provides flexibility in quickly implementing new kernels or kernels from unknown domains
Examples:
x86
Arm
RISC-V
On-Chip Interconnect
Energy-efficient on-chip data movement between different processing elements
Examples:
Crossbar
Network-on-chip
Bus, point-to-point

Fig. 4. An overview of hardware components in DSAs.

problems include reducing the on-chip communication and efficiently mapping the PEs to the NoC [27, 73, 122]. The application trace analysis also provides the volume of data communicated between two kernels. This enables us to exploit optimal mapping and placement of the PEs in the NoC to minimize the on-chip communication latency and energy [187]. In summary, the domain representation analysis is the first and one of the crucial steps in developing DSAs. It acts as a key enabler for hardware development (Section 4), resource management algorithms (Section 5), and the software stack (Section 6). Uhrie et al. [171] use compiler intermediate representations to produce application traces and segment the computational kernels [171]. Then, the data dependencies between the kernels are identified from the traces. However, the limitation here remains the runtime of the tool. To this end, DSAs demand dynamic analysis and transformation techniques (potentially even using just-in-time compilation approaches) to scope applications, and extract the kernels and flow graphs with minimal runtime.

4 HARDWARE ARCHITECTURE AND DESIGN

This section focuses on approaches to design highly energy efficient hardware for the acceleration candidates identified by domain representation approaches discussed in Section 3. Customized hardware designs allow architects to exploit parallelism and orchestrate a highly optimal data flow to maximize energy efficiency for domain-specific computations [138]. We cover the components in hardware architectures in the context of DSAs, as shown in Figure 4.

Fixed-function accelerators. A fixed-function accelerator is designed to implement one specific function, which may be parameterized for different input sizes. Hardware accelerators target the kernels identified in Section 3 to maximize energy efficiency [39]. Fixed-function designs offer the highest benefits since architects highly optimize them for specific operations. Their efficiency begins to diminish when they have to generalize for multiple types of computation. The autonomous driving pipeline in the work of Lin et al. [109] identifies that deep neural network inference and image feature extraction operations consume 95% of the computation time. Accelerating these two operations using fixed-function accelerators provides a 93× overall improvement in end-to-end latency. Another recent autonomous driving pipeline [10] accelerates 2D convolution, FFT, Viterbi decoding, object detection, and tracking operations using fixed-function accelerators. Similarly, a recent image processing pipeline [116] enables 133× speedup by accelerating linear algebraic matrix operations, feature detection, and tracking operations using fixed-function accelerators.

Specialized processors. A specialized processor, also called a *special-purpose core*, a *specialized core*, or a *domain-specific accelerator*, includes any dedicated hardware design that implements more than just one specific function (most often implements a group of similar functions). Developing a dedicated accelerator for each computationally intensive kernel can be highly time consuming. A specialized processor that accelerates multiple kernels reduces the design effort by facilitating extensive design reuse. The most commonly used specialized processors are **coarse-grain reconfigurable arrays (CGRAs)**, systolic arrays, general-purpose graphics processing units, and FPGAs [45, 98, 126, 182, 189]. All of these computing philosophies employ SIMD execution and enable high levels of parallelism or a combination of both [37, 38].

The similarity of operations in domain applications allows us to benefit from specialized processors. For example, a recent CGRA design [45] supports a range of image processing kernels, namely gradient (medical imaging), convolution (digital signal processing), and the Sobel edge detection algorithm (image processing). Similarly, the domain-adaptive systolic array processor presented in the work of Chen et al. [41] accelerates several kernels in wireless communication and linear algebra computation. The machine learning and AI domains benefit substantially from specialized processor implementations. For example, a reconfigurable architecture targeted for convolutional neural networks is presented in the work of Tu et al. [169]. It achieves one to two orders of magnitude higher performance than state-of-the-art designs on AlexNet [102], VGG-19 [154], GoogLeNet [163], and the ResNet-50 [84] models. GPUs are also widely used in neural network applications. They are used to train neural networks for autonomous driving applications, such as in the Tesla autopilot system [88], and execute neural network inference on Nvidia Drive PX2 [115]. FPGA-based parallel implementations have accelerated a wide range of operations, from matrix multiplications [52] to Fourier transforms [149], and are currently extensively deployed to enable high-throughput and low-latency neural network inference computations [159, 190]. The design of specialized processors carefully considers the range of kernels to be supported, which in turn depends on the applications. At this juncture, specialized processors are transformed into DSAs by incorporating domain knowledge to tailor the hardware design to the domain kernels [85, 86].

General-purpose processors. A DSA with only fixed-function and special-purpose processors becomes unusable in two scenarios: (1) tasks in domain applications that cannot be implemented on fixed-function accelerators and specialized processors, and (2) applications outside the target domain. In both scenarios, customized hardware leaves little flexibility on the table to execute incompatible tasks. For this reason, using general-purpose cores in DSAs provides flexibility for other domain applications, albeit with lower energy efficiency [101].

On-chip interconnect. Accelerator-rich designs, such as DSAs, experience significant data movement between the different PEs and can account for up to 40% of the total execution time [153]. Therefore, it is crucial to deploy efficient on-chip communication hardware such that data is moved in a highly energy efficient manner [87, 170]. Prominent interconnect solutions include point-to-point networks, bus-based interconnects, crossbar-based interconnects, and NoCs [118, 158]. NoCs provide ultra-low latency (in the order of tens of nanoseconds for up to 16 PEs) compared to crossbar interconnects (in the order of hundreds of nanoseconds) at the expense of chip area and power consumption [2, 118]. The target domain for the particular DSA and the performance, area, power, and energy constraints play a crucial role in choosing between the different solutions to enable efficient on-chip data movement.

The hardware designs and architectures presented in this section achieve superior energy efficiency for their target applications. However, this process requires extensive design skills, time, and expertise. Agile development techniques and high-level synthesis approaches are a giant leap forward in the automatic generation of custom hardware [72, 119]. However, there is still a significant gap in automatically generating specialized hardware while being aware of other hardware units, communication and coherency models, and application constraints such as latency, power, energy, throughput, and bandwidth.

5 RESOURCE MANAGEMENT IN DSAs

DSAs offer multiple alternative PEs to execute tasks, such as general-purpose cores, hardware accelerators, and specialized processors. To exploit the potential of DSAs, one of the most critical aspects remains the ability to efficiently utilize the available PEs for task execution [101, 175, 187]. This section discusses the resource management aspects of DSAs, key bottlenecks, and outstanding research problems.

The techniques fall broadly into two categories: (1) static (or design-time) and (2) dynamic (or runtime) techniques. Static algorithms utilize the design-time information to manage the resources [11, 165, 167]. These algorithms can provide optimal or heuristic solutions since they are not bounded by computation and latency constraints [193]. Static approaches cannot access runtime information and are inefficient in several scenarios [101]. DSAs inherently support several simultaneous applications that could demand a substantial amount of system resources. Static algorithms may suffice in limited application-specific scenarios; however, DSAs require efficient runtime resource management techniques. Although several static and dynamic approaches have been proposed previously [24, 70, 99, 141, 155], the following fundamental challenges drive the research need for novel dynamic techniques that target DSAs:

- *Heterogeneity:* PEs with different power and performance characteristics for various applications require algorithms to evaluate all valid execution alternatives to obtain the optimal solution. Considering the characteristics of all heterogeneous PEs at runtime makes the resource management problem complex.
- *Streaming arrivals:* Most applications (e.g., video/signal processing, autonomous driving, radar systems) continuously perform identical operations on streaming data frames. The complexity lies in efficiently managing the resources when randomly arriving frames overlap with currently executing and pending tasks from previous frames.
- *Concurrent applications:* SoCs execute several applications simultaneously. Resource management techniques must recognize the divergent application characteristics and satisfy their compute requirements, performance, power, and deadline constraints.

The type of applications and the choice of hardware components in the DSA play a critical role in developing resource management techniques. Therefore, the outputs of domain representation and hardware design are critical inputs to the study of task scheduling techniques (Section 5.1), voltage-frequency scaling policies (Section 5.2), and other aspects (Section 5.3), as shown in Figure 5.

5.1 Task Scheduling and Mapping

DSAs typically execute several streaming applications simultaneously [28, 101]. Task scheduling algorithms assign tasks to the PEs on the DSA to optimize performance (e.g., execution time), power, and energy consumption objectives. Static schedulers utilize only design-time information, whereas dynamic scheduling techniques exploit the runtime information to make effective decisions [167, 175]. Since task scheduling in heterogeneous SoCs is NP-complete, finding the optimal solution is not feasible at runtime for practical problem sizes [26, 172].

Task scheduling algorithms are broadly classified into (1) optimization-based approaches, (2) heuristic techniques, and (3) machine learning based schedulers [120, 167, 193]. Optimization-based approaches formulate the task scheduling problem using objective functions and constraints that describe the applications and the computing platform [165]. The complexity of optimization-based approaches prohibits their use in dynamic scheduling scenarios [193]. Similarly, simulated

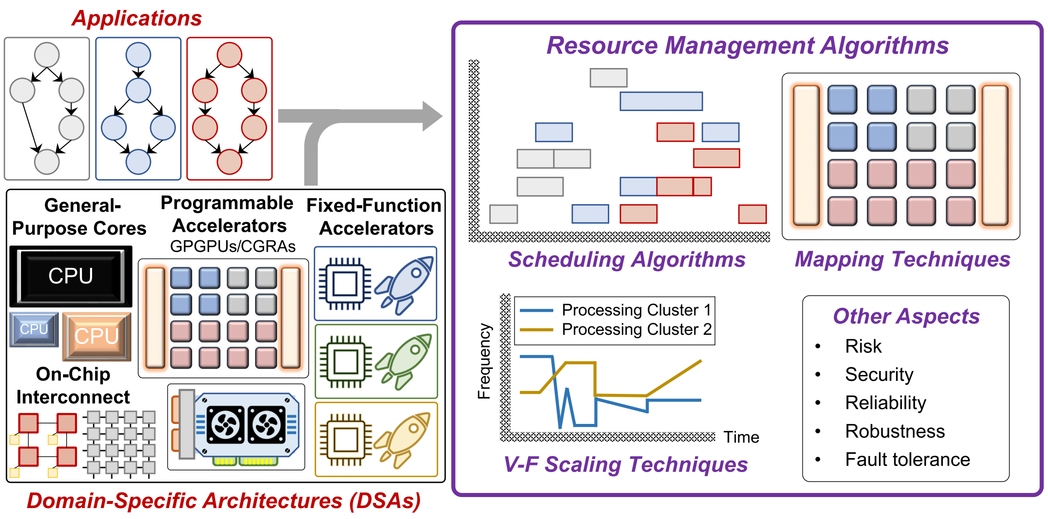

Fig. 5. The key areas of focus in DSA resource management techniques that interface with the domain applications and DSA hardware. The domain representation outputs and hardware design decisions are used to design task scheduling, voltage-frequency scaling, and other resource management techniques.

annealing and genetic algorithm based scheduling algorithms, such as the ones proposed by Jin et al. [94], also suffer from excessive runtime overheads. The high overheads make them impractical for runtime applications. Heuristic schedulers address this challenge and are extensively studied in the literature. HEFT (heterogeneous earliest finish time) is a well-known static list scheduling heuristic technique for heterogeneous platforms [167]. HEFT inspired a family of schedulers that optimize for different objectives [11, 26, 112]. Other static heuristic schedulers such as minimum execution time, tabu search, and genetic and simulated annealing algorithms are presented in the work of Braun et al. [29]. Heuristic schedulers are tailored to sub-optimal objectives and do not generalize well. Recently, machine learning based approaches have been deployed for task scheduling. The DeepRM [120] framework presents the use of **reinforcement learning (RL)** with deep neural networks to perform scheduling in data clusters. The Decima [121] framework uses RL with graph neural networks for the same cluster scheduling problem. RL suffers from the complexity of the reward function design and convergence times. Hence, the imitation learning based approach presented in the work of Krishnakumar et al. [101] poses scheduling in DSAs as a classification problem; then, it uses supervised learning techniques to approximate an Oracle created offline. This technique generalizes to several objectives and dynamic scenarios but does not consider the application deadlines [193, 194] and real-time constraints [48]. Therefore, there is a strong need for a combination of optimality, deadline awareness, and low runtime complexity in task scheduling algorithms for DSAs.

The task scheduling algorithms discussed previously choose between the PEs such as CPUs, hardware accelerators, and specialized processors for a particular task at runtime. However, deciding where to map the different tasks within the SIMD/execution units and communicate between them in a specialized processor is still an outstanding problem. Specifically, the specialized processors (discussed in Section 4) comprise several small execution units, which we refer to as sub-PEs. These processors support the simultaneous execution of several tasks [41, 75]. The factors that influence the task mapping within these processors include the number of sub-PEs, latency, communication information, memory requirements, power, and energy constraints for each simultaneous task [111]. Optimization-based and heuristic techniques (e.g., modulo scheduling techniques) statically generate compile-time mappings [43, 177, 184, 192]. A recent technique [178] introduces dynamic configuration to adapt a systolic array at runtime based on the neural network sizes. A more general formulation of the sub-PE mapping problem can be found in the work of Chou et al. [44]. The critical challenge remains to generate a completely automatic and optimal mapping at runtime with the least possible latency and energy consumption overheads.

5.2 Dynamic Thermal-Power Management Techniques

State-of-the-art PEs and cores support multiple **voltage and frequency (V-F)** levels. Another critical aspect in exploiting the potential of DSAs is optimally selecting these power states at runtime [108, 152]. The V-F levels of PEs play a primary role in determining power and performance, and the power consumption determines the temperature of the PEs [25]. For example, using the highest V-F levels to maximize performance increases power consumption and, in turn, the temperature [81]. Furthermore, portions of the SoC may be placed in different levels of sleep states where they are partially or entirely powered off to save energy and control temperature [180]. Like task scheduling, V-F selection for the cores is also NP-complete [164]. Therefore, efficient **dynamic thermal-power management (DTPM)** techniques are essential to utilize DSAs efficiently while maintaining the chip temperature within limits [130].

The DTPM techniques also fall into categories similar to the task scheduling algorithms, namely optimization-, heuristic-, and machine learning based techniques. The extensive use of heterogeneous **multiprocessor systems-on-chip (MPSoCs)** in battery- and energy-constrained systems has attracted substantial research in this domain. The heuristic techniques proposed by Han et al. [80] and Reddy et al. [146] use the difference between achieved and target metrics to adjust the frequencies. The approach presented by Moazzemi et al. [128] combines the benefits of traditional control-theoretic approaches and heuristics to develop a lightweight and efficient frequency scaling policy. Recent approaches presented in other works [51, 117, 152] use machine learning to train policies to determine the optimal operating frequency and generalize to unseen workloads. Temperature management on heterogeneous SoCs is also critical as the on-die power density critically increases [25, 50, 150]. Current approaches for power and thermal management techniques focus on homogeneous and heterogeneous CPU cores and also on GPUs, comprehensively discussed in the work of Pasricha et al. [137]. A few techniques consider hardware accelerators in their power management policies [53, 185]. However, DSAs demand novel techniques that consider all types of hardware accelerators and specialized cores since they can significantly contribute to the overall power, energy, and temperature.

5.3 Other Resource Management Research Directions

Although task scheduling, mapping, and DTPM ideas dominate the primary aspects of resource management, modern-day SoCs look at other aspects to satisfy requirements such as reliability and security to meet user expectations and privacy standards. For instance, SoCs and processors from Apple, Intel, and RISC-V include a secure enclave to protect user data when the platform is experiencing security attacks [1, 3, 131]. Instead of relying solely on the secure enclave to protect sensitive data, building security into other aspects enhances the security of the design. DSAs seek adaptation and advancement of ideas from prior work on heterogeneous MPSoCs.

Risk and security. Integrating security into scheduling algorithms and **dynamic voltage frequency scaling (DVFS)** governors is at the expense of chip area, scheduling latency, power and energy overheads, and design complexity. Therefore, low complexity and runtime overheads remain essential requirements of security-aware techniques. Commercial SoCs integrate several

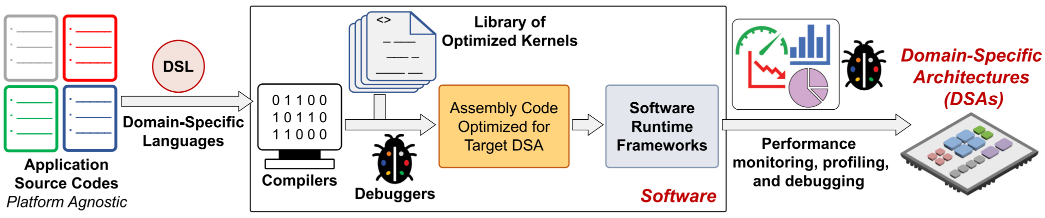
DSL
Domain-Specific Languages
Application Source Codes
Platform Agnostic
01100
10110
11000
Compilers
Library of Optimized Kernels
Debuggers
Assembly Code Optimized for Target DSA
Software Runtime Frameworks
Software
Performance monitoring, profiling, and debugging
Domain-Specific Architectures (DSAs)

Fig. 6. The software stack processes the application source codes expressed in domain-specific or traditional languages. The software tools include specialized compilers and a pre-compiled library of highly optimized kernels to execute applications in runtime frameworks on the target DSA.

third-party IPs to promote design reuse and improve the design cycle [136]. However, using external IPs has the severe risk of untrusted designs, leading to security flaws. To this end, a multi-dimensional optimization approach improves the security of the MPSoC through task scheduling with negligible impact on performance and no additional hardware cost [110]. In this approach, task duplication and isolation are the two techniques that aid in detecting hardware Trojans in the presence of third-party IPs. Recent literature has shown that security attacks can extract sensitive data by exploiting the temperature patterns on the chip [91]. Indeed, the ThermalAttackNet [49] discusses the potential of DVFS governors in avoiding the detection of stored passwords using on-chip temperature patterns. Therefore, integrating such techniques into resource management algorithms improves the security of DSAs.

Reliability and robustness. With the increasing use of SoCs in safety-critical applications (e.g., autonomous driving, avionics, and medical applications), there is a critical emphasis on reliable and robust computing [103, 186]. The chip temperature plays a critical role in the mean time to failure since it directly impacts the metal fatigue. A tradeoff between power consumption and reliability (in the mean time to failure) is explored by estimating the failure rate at a given chip temperature in the work of Rosing et al. [148]. The DVFS technique proposed in this article integrates a reliability metric into its optimization problem to increase the mean time to failure. Furthermore, differential aging of cores in an SoC results in certain cores failing sooner than the other counterparts. To address this challenge, the scheduling approach presented by Huang et al. [90] includes the mean time to failure in the objective function along with deadline constraints. Although the preceding techniques focus on prolonging the time to failure, task scheduling techniques also provide reliable and robust decisions in the presence of system faults [54, 89]. For space-based applications, state-of-the-art approaches must provide reliable and robust decision in high-radiation environments [54, 58, 139]. In summary, emerging reliability and robustness requirements demand resource management approaches for DSAs to take them into consideration to enhance usability.

6 SOFTWARE DEVELOPMENT

Customized hardware designs are typically notoriously hard to program. Hence, one of the fundamental research challenges is maximizing the performance and energy efficiency of DSAs while relieving the end users from the platform-specific details. DSA software infrastructures aim to facilitate end-user application development and make it agnostic to the hardware platform using the tools and frameworks summarized in Figure 6.

6.1 Domain-Specific Languages

Domain-specific languages (DSLs) are designed for particular application domains. DSLs enhance expressiveness and programmer productivity to generate highly optimized code implementations to improve platform performance and portability [125]. They offer high programming accessibility, reduced code complexity, and improved programmer productivity through predefined domain-specific abstractions [30]. DSL-based compilers utilize these abstractions to enable targeted optimizations [143]. The Delite DSL [160] embeds into Scala (a general-purpose programming language) and translates code constructs to programming models such as C++, CUDA, and OpenCL. Halide enables high-performance image processing through domain-specific constructs and generates efficient implementations for x86, ARM, and GPUs [145]. The Graphit DSL for graph-based computations generates fast and optimized implementations by considering the graph structure, algorithm, and the underlying hardware platform [191]. The **heterogeneous parallel virtual machine (HPVM)** DSL and compiler infrastructure makes strong strides toward DSAs as it targets highly heterogeneous and parallel systems [100]. The use of DSLs eases the design and programming of DSAs, aiding the effort to maximize its performance and energy efficiency [18, 119].

6.2 Compilers and Debuggers

Programming heterogeneous SoCs, especially DSAs, is a monumental challenge for the following reasons: (1) hardware heterogeneity, (2) identifying parallelism in applications and hardware, and (3) memory hierarchy and data movement [86, 100, 161, 171]. The HPVM compiles applications into data flow graphs and develops a representation that exploits the parallelism of the underlying hardware [100]. The framework presented in the work of Xiao et al. [187] includes a compiler that constructs flow graphs from applications using the low-level virtual machine intermediate representation [106]. Then, the framework identifies the optimal number and type of resources to maximize energy efficiency while balancing the computation and on-chip communication costs. The runtime framework in the work of Mack et al. [113] also includes a compilation step that maps application code to DSA platforms. All of these compilers exploit the task- and data-level parallelism inherent in the applications and the hardware. Furthermore, debuggers that improve the code and platform debug capabilities enhance the usability of DSAs.

6.3 Kernel Library

The performance of kernels and applications strongly depends on the programmer's specific implementation. For instance, the execution times of the Fourier transform operation using the GSL [64], FFTW [60], MKL [179], and FFTPACK [129] libraries vary by up to three orders of magnitude [65]. DSAs include a library that contains highly optimized implementations of domain-specific kernels to improve performance. The compiler identifies kernels in the application source code and substitutes them with the optimized implementations from the library [47, 107]. Therefore, a flow that integrates a kernel library that swaps the kernels at compile-time/runtime maximizes performance and reduces the programmer's burden.

6.4 Performance Monitors and Profilers

DSAs strive to keep the end users oblivious to the underlying platform architecture such that architecture expertise and hardware knowledge are not expected from them [74]. However, end users still like to analyze the performance of their applications and evaluate the inefficiencies and bottlenecks in the applications. To this end, the performance monitoring unit, *perf*, and performance application programming interface are some of the tools that abstract the behavior of the CPU execution pipeline (e.g., control unit, cache, and memory) into key performance indicators [152]. These tools are primarily applicable for general-purpose processor-based MPSoC systems. DSAs contain various PEs; hence, the profiling tools require adaptation to obtain performance indicators of hardware accelerators and specialized processors. The performance **application programming**

interface (API) infrastructure is extended to hardware accelerators and validated for a prototype implemented on Xilinx Zynq FPGA [161]. Nvidia provides profiling tools to collect a variety of performance counters helpful for general-purpose graphics processing unit applications [132, 140]. CPUs, GPUs, and other PE types comprise their own performance monitoring mechanisms. Likewise, DSAs need a unified framework that aggregates performance indicating counters from all PEs to serve two essential purposes: (1) enable resource management algorithms to make smarter decisions based on the performance counters, and (2) assist programmers in analyzing application performance and bottlenecks [74].

6.5 Runtime Frameworks

All PEs in DSAs, including general-purpose processors, specialized cores, and fixed-function accelerators, expose APIs for programming them. End users want their existing applications to execute in DSAs with no or minimal modifications and developmental effort [113]. Moreover, the end users are agnostic to the architecture and the type of hardware components. Therefore, there is a critical need for runtime frameworks that use the application source code and execute them on the heterogeneous hardware using state-of-the-art resource management techniques [17]. Runtime frameworks bridge the hardware components, resource management algorithms, and domain representation techniques in DSAs. During application execution, they identify the domain-specific kernels, allocate them to the PEs, and run them on the energy-efficient hardware. Software runtime frameworks also integrate the compiler, kernel library, and profiler aspects to improve performance and user productivity. For example, the StarPU framework [17] schedules applications to CPU-GPU systems. It exploits the inherent heterogeneity and parallelism to achieve better performance than state-of-the-art systems. The userspace runtime framework presented in the work of Mack et al. [114] supports plug-and-play selection of scheduling algorithms for application execution in accelerator-rich DSAs.

7 DESIGN AND EVALUATION FRAMEWORKS FOR RAPID DESIGN

Heterogeneous architectures, particularly DSAs, integrate several in-house and third-party hardware components. This integration increases the design complexity, leading to verification and physical implementation challenges [35, 162]. The DSA design choices are evaluated by several objectives, such as power, performance, energy, and throughput [23, 79]. The shrinking time-to-market requirements pose tight timelines for chip designers. The stringent time requirements demand methodologies, strategies, and evaluation tools for rapid DSA design and development.

7.1 Design Methodologies and Strategies

Design methodologies and strategies reduce human effort, design-time, and development costs [62, 147]. The ideas that are critical for DSAs to minimize the time-to-market, reduce design costs, improve developer productivity, and build efficient designs are listed next:

- *Design modularity and reuse:* Modularity allows smaller pieces of the design to be developed independently. Furthermore, the modules can be substituted by newer versions with minimal changes to the rest of the design [119]. Modularity also helps with extensive design reuse for frequently used logic and hardware blocks [20].
- *Open instruction set architectures:* The influence of open-source software paved the way for open **instruction set architectures (ISAs)** [86]. The RISC-V ISA is one such prominent example [76]. Open ISA promotes collaboration, design reuse, affordability, and, most importantly, security [4]. Proprietary ISAs obscure the more delicate details within a closed set

Table 1. Overview of the Types of Evaluation Frameworks and Their Offerings in DSA Realization

	High-Level Simulation Frameworks	**Cycle-Accurate Simulation Frameworks**	**Emulation Frameworks**
Evaluation Accuracy	Moderate	High	Highest
Speed of Evaluation	Fastest	Slow	Fast
Ease of Flexibility	Highest	Moderate	Low
Evaluation of Resource Management Algorithms	Fast	Slower	Slowest
Modeling Technique	Analytical and approximate models	Fine-grained models (and/or) real implementations	Real implementations
Design Space Exploration	Fast	Slower	Slowest
Software Development	Not supported	Limited support	Full support
Firmware Development	Not supported	Not supported	Full support
Functional Validation	Not supported	Not supported	Supported

of individuals, whereas open ISAs promote security by allowing the community to evaluate the risks and robustness [14].

- *Open-source hardware and tools:* Open-source hardware encourages design reuse and exploiting pre-validated hardware. It allows the community to collectively develop superior hardware, like the success of the Linux kernel. Significant emphasis is also being placed on developing open-source hardware design tools [4, 9, 33, 92].
- *Agile hardware development flow:* The agile methodology empowers small teams (fewer than 10 members) and realizes work in short sprints (2–4 weeks). The sprints are thoroughly planned and evaluated by regular reviews [144]. The remarkable success of agile flows in software development strongly influences its adoption in hardware development [18]. Although some software concepts are not directly applicable to hardware due to the differences in timelines, the hardware community welcomes the approach with adaptations to reduce the DSA time-to-market.

7.2 Evaluation Frameworks

The DSA design complexity and time-to-market requirements motivate the need for rapid and efficient design space exploration [173]. The complexity of these architectures also requires comprehensive verification techniques. Functionally validating designs before tape-out reduce the chance of post-silicon failures and improve the time-to-market. Modern SoCs demand early firmware and software development to further shrink the design cycles, typically before tape-in. To this end, emulation platforms enable functional validation and early software development. They can reduce the software bring-up time to less than 24 hours within the availability of the chip. The goals of simulation and emulation frameworks and their support in realizing DSAs are presented in Table 1. This section discusses the initial work targeting this research direction, the current gaps, and the efforts required to bridge this gap.

Simulation frameworks. Simulation and modeling of full systems can be broadly classified into (1) specification-level modeling, (2) transaction-level modeling, and (3) cycle-accurate modeling [61]. Cycle-accurate simulators use precise behavioral models to obtain accurate estimates, but they incur prohibitively long runtimes to perform rapid design space explorations [16, 176]. High-level simulators use approximate and analytical models to trade off estimation accuracy for speed [12, 16]. Gries [77] discusses the necessity, advantages, and methods for fast design space exploration in MPSoCs. The SpecC and SystemC languages have fueled the growth of transaction-level modeling based system design and exploration [63, 71, 78]. DSLs are an abstraction layer between users and the simulation tools to provide platform information [56]. Aspen, a DSL, formally specifies the application's functional behavior and an abstract platform model [157]. The model and application behavior are then utilized to project runtime and roofline charts, and evaluate performance. An abstract model to explore dynamic mapping strategies for NoC-based MPSoCs in [34] achieves a 91% reduction in simulation compared to RTL-based models. In addition, a large body of work further accelerates design space exploration of MPSoCs on FPGA [7, 8, 22].

Rapid DSA design exploration using high-level simulation has been explored in other works [12, 175]. Both simulators are examples of host-compiled simulation frameworks [69]. They bridge the gap between high-level models and real implementations by integrating profiling-based timing estimates. The STOMP simulator facilitates easy evaluation of scheduling policies for DSAs in the presence of real-time applications and deadline constraints [175]. The DS3 simulator allows users to evaluate scheduling algorithms, DVFS policies, and SoC configurations with metrics such as power, execution time, and energy consumption [12].

Emulation platforms. Emulation frameworks overcome the limitation of simulators by enabling functional verification, and early software and firmware development [22, 67, 124]. The emulation frameworks are broadly classified into virtual model-based emulators and FPGA-based frameworks that rely on the actual implementation [42]. It is worth noting that only FPGA-based frameworks allow functionality validation since they use the actual implementation, as compared to representative models in virtual platforms.

The emergence of SystemC and transaction-level-based modeling also significantly enhanced the development of virtual emulation frameworks [40]. The quick emulator (QEMU) deploys abstract models of the computing elements and transaction-level models for its interactions with the rest of the system [21]. QEMU also enables developers to bring up a variety of guest operating systems and execute applications on the CPU through dynamic binary translations [19]. Along these lines, ARM provides fast models that are accurate and representative models of their IPs such as CPUs, interconnects, sub-systems, and other peripheral components [174]. Fast models allow the bring-up of the Linux OS, and programmers to develop software, firmware, and applications.

FPGA-based frameworks also improve task scheduling and DTPM policies by utilizing more realistic estimates in MPSoCs [15, 130]. An MPSoC-based sensor- and actuator-rich cyber-physical SoC is prototyped in the work of Sarma and Dutt [151], enabling hardware and software co-design. Other frameworks for MPSoCs are presented and discussed in the work of Khamis et al. [97] and Kurth et al. [101]. The MPSoC-based frameworks must be adapted for DSAs by integrating the other components, such as specialized cores, hardware accelerators, and on-chip interconnects. In that direction, the FPGA-based user-space emulation framework in the work of Mack et al. [114] integrates hardware and software to evaluate resource management policies for DSAs. Although this work is an initial step in the DSA direction, frameworks that scale to the entire design are critical to bridge the gap between the requirements and state-of-the-art approaches.

8 INTERACTION BETWEEN THE DSA RESEARCH DIRECTIONS, INSIGHTS, AND OPEN CHALLENGES

8.1 Interactions between the Research Directions

This section discusses the interactions between the research directions presented so far, as outlined in Figure 7. Domain representation efforts analyze the applications and present computational kernels that are potential candidates for implementation in special-purpose hardware. The hardware architecture and design exploit this information to develop customized and efficient processing for these potential candidates using either fixed-function or specialized accelerators. Hardware design techniques also leverage the data dependencies between the kernels in applications to

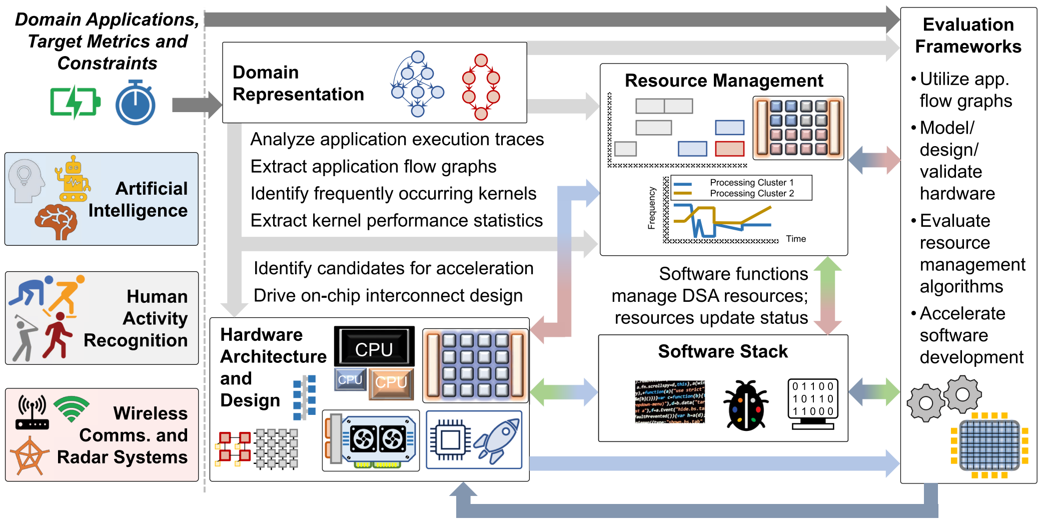
Domain Applications, Target Metrics and Constraints
Artificial Intelligence
Human Activity Recognition
Wireless Comms. and Radar Systems
Domain Representation
Analyze application execution traces
Extract application flow graphs
Identify frequently occurring kernels
Extract kernel performance statistics
Identify candidates for acceleration
Drive on-chip interconnect design
Hardware Architecture and Design
CPU
Resource Management
Processing Cluster 1
Processing Cluster 2
Frequency
Time
Software functions manage DSA resources; resources update status
Software Stack
Evaluation Frameworks
• Utilize app. flow graphs
• Model/ design/ validate hardware
• Evaluate resource management algorithms
• Accelerate software development

Fig. 7. Interactions between the research directions for the realization of DSAs. The kernel and flow graph information of domain applications and target metrics drive the hardware architecture and design of PEs. The application and hardware PE information is exploited by the resource management techniques. The DSA configurations are evaluated for functionality and performance using simulators, emulation frameworks, and the software stack.

design an appropriate on-chip communication network, such as bus, point-to-point network, and NoC. Similarly, the domain analysis provides design-time and runtime information to the resource management algorithms. Design-time information includes the kernel characteristics and their interactions that affect latency, execution time, and communication volumes. Resource management algorithms exploit this information offline to deploy targeted scheduling and power management techniques for the chosen PEs. The domain-specific information in the applications helps narrow down the vast design space. Simulation frameworks perform rapid design space explorations and systematically evaluate resource management algorithms. The domain representation techniques and software stack share similar tools and infrastructure, such as compilation and performance profiling APIs. They target specific hardware by providing the relevant compiler and API support. Finally, emulation frameworks accelerate software development, enable performance evaluation, and facilitate functional validation to improve the time-to-market for DSAs.

8.2 Insights and Open Challenges

DSAs are making solid advances toward becoming the preferred choice for future computing systems. DSA design and development efforts require significant attention as the algorithms, design methodologies, and tools evolve. Furthermore, the subtle interaction between the different DSA research aspects requires substantial research focus. The ever-lasting pursuit of maximizing performance and energy efficiency while minimizing the cost and design effort leaves the following open research questions:

- How can we reduce the time required to generate application traces, scope them and extract the flow graphs?
- Can we perform DSA hardware-aware application code compilation using state-of-the-art and just-in-time compilation techniques?

- Can we automatically generate highly optimized and specialized hardware based on high-level requirements, such as performance, power, and throughput?
- How can we architect easily programmable and flexible yet highly specialized hardware?
- Can we design light-weight and near-optimal resource management algorithms considering all application requirements, such as performance, power consumption, energy efficiency, and deadlines?
- Can we explore preemption-based resource management techniques with hardware accelerators (that do not allow context switching) to address real-time needs?
- How can we automatically generate software support for custom-designed hardware accelerators and reduce the development time?
- How can we accurately and quickly calibrate high-level simulators with pre-silicon data or real hardware, speed up cycle-accurate simulations, and reduce the development times for emulation frameworks?
- How can we embed security and privacy into all DSA design aspects and components?
- How can we seamlessly integrate the different DSA research directions and tools to maximize performance and energy efficiency with minimum intervention from users and developers?

A few challenges involved in DSA design listed in the different sections are compiled here. These and similar questions demand innovation and research for performant and energy-efficient DSA-based computing systems.

9 CONCLUSION

The slowdown of Moore's law and Dennard scaling has limited the power and performance gains obtained with the evolution of technology process nodes over the years. There is a strong need for innovation in several aspects, such as ISA, microarchitecture, algorithms, hardware, and software, to achieve drastic improvements in energy efficiency. Beyond these conventional approaches, DSAs promise to achieve superior energy efficiency by combining general-purpose cores and hardware accelerators for applications in a target domain. Furthermore, DSAs integrate sophisticated runtime resource management algorithms and a software stack to maximize performance and energy efficiency.

This survey discussed various research directions and challenges in designing and developing DSAs. We also presented some promising approaches and gaps to be addressed to develop and use DSAs quickly. As we witness tremendous innovation, DSAs are expected to contribute to the quest for higher performance and energy efficiency in future computing systems.

ACKNOWLEDGMENTS

The views and conclusion contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

REFERENCES

[1] Apple. [n.d.]. Apple Secure Enclave. Retrieved May 15, 2022 from https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web.

[2] Cadence. [n.d.]. ARM CoreLink Interconnects Whitepaper. Retrieved May 15, 2022 from https://ip.cadence.com/uploads/251/white-paper-interconnect-solutions-debugging-issues-advanced-ARM-CoreLink-pdf.

[3] ARM. [n.d.]. ARM TrustZone. Retrived May 15, 2022 from https://developer.arm.com/documentation/PRD29-GENC-009492/c/TrustZone-Hardware-Architecture.

[4] Google. [n.d.]. Google's Thrust Towards Open-Source Hardware. Retrieved May 15, 2022 from https://opensource.googleblog.com/2019/05/google-fosters-open-source-hardware.html.
[5] Aakash Jani. 2022. Year in Review: PC Processors Adopt Hybrid CPUs. Retrieved May 15, 2022 from https://www.techinsights.com/blog/year-review-pc-processors-adopt-hybrid-cpus.
[6] Retrieved May 15, 2022 from https://futurenetworks.ieee.org/images/files/pdf/FirstResponder/Tom-Rondeau-DARPA.pdf.
[7] Siemens. [n.d.]. Veloce2 Emulator. Retrieved May 15, 2022 from https://www.mentor.com/products/fv/emulation-systems/veloce.
[8] Synopsys. [n.d.]. ZeBu Server 4. Retrieved May 15, 2022 from https://www.synopsys.com/verification/emulation/zebu-server.html.
[9] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, et al. 2019. Toward an open-source digital flow: First learnings from the OpenROAD project. In *Proceedings of the 56th Annual Design Automation Conference*. 1–4.
[10] Aporva Amarnath, Subhankar Pal, Hiwot Tadese Kassa, Augusto Vega, Alper Buyuktosunoglu, Hubertus Franke, John-David Wellman, Ronald Dreslinski, and Pradip Bose. 2021. Heterogeneity-aware scheduling on SoCs for autonomous vehicles. *IEEE Computer Architecture Letters* 20, 2 (2021), 82–85.
[11] Hamid Arabnejad and Jorge G. Barbosa. 2013. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (2013), 682–694.
[12] Samet Arda, Anish Krishnakumar, Ahmet Alper Goksoy, Joshua Mack, Nirmal Kumbhare, Anderson Luiz Sartor, Ali Akoglu, Radu Marculescu, and Umit Y. Ogras. 2020. DS3: A system-level domain-specific system-on-chip simulation framework. *IEEE Transactions on Computers* 69, 8 (2020), 1248–1262.
[13] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, et al. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report No. UCB/EECS-2006-183. EECS Department, University of California, Berkeley.
[14] Krste Asanović and David A. Patterson. 2014. *Instruction Sets Should Be Free: The Case for RISC-V*. Technical Report No. UCB/EECS-2014-146. EECS Department, University of California, Berkeley.
[15] David Atienza, Pablo G. Del Valle, Giacomo Paci, Francesco Poletti, Luca Benini, Giovanni De Micheli, Jose M. Mendias, and Roman Hermida. 2008. HW-SW emulation framework for temperature-aware design in MPSoCs. *ACM Transactions on Design Automation of Electronic Systems* 12, 3 (2008), 1–26.
[16] Rabie Ben Atitallah, Smail Niar, Samy Meftali, and Jean-Luc Dekeyser. 2007. An MPSoC performance estimation framework using transaction level modeling. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*. 525–533.
[17] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. 2011. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience* 23, 2 (2011), 187–198.
[18] Rick Bahr, Clark Barrett, Nikhil Bhagdikar, Alex Carsello, Ross Daly, Caleb Donovick, David Durst, et al. 2020. Creating an agile hardware design flow. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. 1–6.
[19] Daniel Bartholomew. 2006. QEMU: A multihost, multitarget emulator. *Linux Journal* 2006, 145 (2006), 3.
[20] Amir H. Behzadan, Brian W. Timm, and Vineet R. Kamat. 2008. General-purpose modular hardware and software framework for mobile outdoor augmented reality applications in engineering. *Advanced Engineering Informatics* 22, 1 (2008), 90–105.
[21] Fabrice Bellard. 2005. QEMU, A fast and portable dynamic translator. In *Proceedings of the USENIX Annual Technical Conference: FREENIX Track*, Vol. 41. 10–5555.
[22] Giovanni Beltrame, Luca Fossati, and Donatella Sciuto. 2009. ReSP: A nonintrusive transaction-level reflective MPSoC simulation platform for design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 12 (2009), 1857–1869.
[23] Mehmet E. Belviranli and Jeffrey S. Vetter. 2019. FLAME: Graph-based hardware representations for rapid and precise performance modeling. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'19)*. 1775–1780.
[24] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Scale Integration (VLSI) Systems* 8, 3 (2000), 299–316.
[25] Ganapati Bhat, Gaurav Singla, Ali K. Unver, and Umit Y. Ogras. 2017. Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 3 (2017), 544–557.
[26] Luiz F. Bittencourt, Rizos Sakellariou, and Edmundo R. M. Madeira. 2010. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Proceedings of the Euromicro Conference on Parallel, Distributed, and Network-Based Processing*. 27–34.

[27] Behzad Boroujerdian, Ying Jing, Devashree Tripathy, Amit Kumar, Lavanya Subramanian, Luke Yen, Vincent Lee, et al. 2022. FARSI: An early-stage design space exploration framework to tame the domain-specific system-on-chip complexity. *ACM Transactions on Embedded Computing Systems.* Online, June 16, 2022.

[28] Pradip Bose, Augusto Vega, Sarita Adve, Vikram Adve, Sasa Misailovic, Luca Carloni, Ken Shepard, David Brooks, Vijay Janapa Reddi, and Gu-Yeon Wei. 2021. Secure and resilient SoCs for autonomous vehicles. In *International Workshop on Domain Specific System Architecture (DOSSA), in conjunction with IEEE International Symposium on High-Performance Computer Architecture (HPCA).* https://scholar.google.com/scholar?hl=en&as_sdt=0%2C50&q=Secure+and+resilient+SoCs+for+autonomous+vehicles&btnG=.

[29] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau L. Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, et al. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* 61, 6 (2001), 810–837.

[30] Kevin J. Brown, Arvind K. Sujeeth, Hyouk Joong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2011. A heterogeneous parallel framework for domain-specific languages. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques.* 89–100.

[31] Brad Burres, Dan Daly, Mark Debbage, Eliel Louzoun, Christine Severns-Williams, Naru Sundar, Nadav Turbovich, Barry Wolford, and Yadong Li. 2021. Intel's hyperscale-ready infrastructure processing unit (IPU). In *Proceedings of the IEEE Hot Chips 33 Symposium (HCS'21).* 1–16.

[32] Idan Burstein. 2021. Nvidia data center processing unit (DPU) architecture. In *Proceedings of the IEEE Hot Chips 33 Symposium (HCS'21).* 1–20.

[33] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson. 2013. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. *ACM Transactions on Embedded Computing Systems* 13, 2 (2013), 1–27.

[34] Everton A. Carara, Roberto P. De Oliveira, Ney L. V. Calazans, and Fernando G. Moraes. 2009. HeMPS—A framework for NoC-based MPSoC generation. In *Proceedings of the International Symposium on Circuits and Systems.* 1345–1348.

[35] Luca P. Carloni. 2016. The case for embedded scalable platforms. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC'16).* 1–6.

[36] Jeronimo Castrillon, Rainer Leupers, and Gerd Ascheid. 2011. MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs. *IEEE Transactions on Industrial Informatics* 9, 1 (2011), 527–545.

[37] Nagadastagiri Challapalle, Sahithi Rampalli, Makesh Chandran, Gurpreet Kalsi, Sreenivas Subramoney, John Sampson, and Vijaykrishnan Narayanan. 2020. PSB-RNN: A processing-in-memory systolic array architecture using block circulant matrices for recurrent neural networks. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'20).* 180–185.

[38] Nagadastagiri Challapalle, Sahithi Rampalli, Nicholas Jao, Akshaykrishna Ramanathan, John Sampson, and Vijaykrishnan Narayanan. 2020. FARM: A flexible accelerator for recurrent and memory augmented neural networks. *Journal of Signal Processing Systems* 92, 11 (2020), 1247–1261.

[39] Nagadastagiri Challapalle, Karthik Swaminathan, Nandhini Chandramoorthy, and Vijaykrishnan Narayanan. 2021. Crossbar based processing in memory accelerator architecture for graph convolutional networks. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'21).* 1–9.

[40] Amir Charif, Gabriel Busnot, Rania Mameesh, Tanguy Sassolas, and Nicolas Ventroux. 2019. Fast virtual prototyping for embedded computing systems design and exploration. In *Proceedings of Rapid Simulation and Performance Evaluation: Methods and Tools.* 1–8.

[41] Kuan-Yu Chen, Chi-Sheng Yang, Yu-Hsiu Sun, Chien-Wei Tseng, Morteza Fayazi, Xin He, Siying Feng, et al. 2022. A 507 GMACs/J 256-core domain adaptive systolic-array-processor for wireless communication and linear-algebra kernels in 12nm FINFET. In *Proceedings of the 2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits'22).*

[42] Wen Chen, Sandip Ray, Jayanta Bhadra, Magdy Abadir, and Li-C. Wang. 2017. Challenges and trends in modern SoC design verification. *IEEE Design & Test* 34, 5 (2017), 7–22.

[43] S. Alexander Chin and Jason H. Anderson. 2018. An architecture-agnostic integer linear programming approach to CGRA mapping. In *Proceedings of the 55th Annual Design Automation Conference.* 1–6.

[44] Chen-Ling Chou, Umit Y. Ogras, and Radu Marculescu. 2008. Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 10 (2008), 1866–1879.

[45] Jason Cong, Hui Huang, Chiyuan Ma, Bingjun Xiao, and Peipei Zhou. 2014. A fully pipelined and dynamically composable architecture of CGRA. In *Proceedings of the Annual International Symposium on Field-Programmable Custom Computing Machines.* 9–16.

[46] Jason Cong, Vivek Sarkar, Glenn Reinman, and Alex Bui. 2010. Customizable domain-specific computing. *IEEE Design & Test of Computers* 28, 2 (2010), 6–15.

[47] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, et al. 2021. TensorFlow Lite Micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems* 3 (2021), 800–811.

[48] Robert I. Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys* 43, 4 (2011), 1–44.

[49] Somdip Dey, Amit Kumar Singh, and Klaus McDonald-Maier. 2021. ThermalAttackNet: Are CNNs making it easy to perform temperature side-channel attack in mobile edge devices? *Future Internet* 13, 6 (2021), 146.

[50] Somdip Dey, Amit Kumar Singh, and Klaus Dieter McDonald-Maier. 2019. P-EdgeCoolingMode: An agent-based performance aware thermal management unit for DVFS enabled heterogeneous MPSoCs. *IET Computers & Digital Techniques* 13, 6 (2019), 514–523.

[51] Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'16)*. 1–10.

[52] Yong Dou, Stamatis Vassiliadis, Georgi Krasimirov Kuzmanov, and Georgi Nedeltchev Gaydadjiev. 2005. 64-bit floating-point FPGA matrix multiplication. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 86–95.

[53] Sandeep D'Souza and Ragunathan Rajkumar. 2018. CycleTandem: Energy-saving scheduling for real-time systems with hardware accelerators. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'18)*. 94–106.

[54] Laura A. Rozo Duque, Jose M. Monsalve Diaz, and Chengmo Yang. 2015. Improving MPSoC reliability through adapting runtime task schedule based on time-correlated fault behavior. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'15)*. 818–823.

[55] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*. IEEE, Los Alamitos, CA, 365–376.

[56] Roland Ewald and Adelinde M. Uhrmacher. 2014. SESSL: A domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation* 24, 2 (2014), 1–25.

[57] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. 1987. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems* 9, 3 (1987), 319–349.

[58] Farshad Firouzi, Ali Azarpeyvand, Mostafa E. Salehi, and Sied Mehdi Fakhraie. 2012. Adaptive fault-tolerant DVFS with dynamic online AVF prediction. *Microelectronics Reliability* 52, 6 (2012), 1197–1208.

[59] Alcides Fonseca and Bruno Cabral. 2017. Prototyping a GPGPU neural network for deep-learning big data analysis. *Big Data Research* 8 (2017), 50–56.

[60] Matteo Frigo and Steven G. Johnson. 1998. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'98)*, Vol. 3. 1381–1384.

[61] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. 2009. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Science & Business Media.

[62] Daniel D. Gajski, Sanjiv Narayan, Loganath Ramachandran, Frank Vahid, and Peter Fung. 1996. System design methodologies: Aiming at the 100 h design cycle. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 4, 1 (1996), 70–82.

[63] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. 2012. *SpecC: Specification Language and Methodology*. Springer Science & Business Media.

[64] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, Fabrice Rossi, and Rhys Ulerich. 2002. *GNU Scientific Library*. Network Theory Limited.

[65] P. Gambron and S. Thorne. 2020. *Comparison of Several FFT Libraries in C/C++*. Technical Report. STFC.

[66] David Geer. 2005. Chip makers turn to multicore processors. *Computer* 38, 5 (2005), 11–13.

[67] Nicolas Genko, David Atienza, Giovanni De Micheli, and Luca Benini. 2007. Feature-NoC emulation: A tool and design flow for MPSoC. *IEEE Circuits and Systems Magazine* 7, 4 (2007), 42–51.

[68] Pawel Gepner and Michal Filip Kowalik. 2006. Multi-core processors: New way to achieve high system performance. In *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*. 9–13.

[69] Andreas Gerstlauer. 2010. Host-compiled simulation of multi-core platforms. In *Proceedings of the 21st IEEE International Symposium on Rapid System Protyping*. 1–6.

[70] Andreas Gerstlauer, Christian Haubelt, Andy D. Pimentel, Todor P. Stefanov, Daniel D. Gajski, and Jürgen Teich. 2009. Electronic system-level synthesis methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 10 (2009), 1517–1530.

[71] Frank Ghenassia (Ed.). 2005. *Transaction-Level Modeling with SystemC*. Vol. 2. Springer.

[72] Davide Giri, Kuan-Lin Chiu, Guy Eichler, Paolo Mantovani, and Luca P. Carloni. 2021. Accelerator integration for open-source SoC design. *IEEE Micro* 41, 4 (2021), 8–14.

[73] Davide Giri, Paolo Mantovani, and Luca P. Carloni. 2018. NoC-based support of heterogeneous cache-coherence models for accelerators. In *Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip (NoCS'18)*. 1–8.

[74] Daniel S. Green. 2018. *Heterogeneous Integration at DARPA: Pathfinding and Progress in Assembly Approaches*. DARPA.

[75] Oded Green, Robert McColl, and David A. Bader. 2012. GPU merge path: A GPU merging algorithm. In *Proceedings of the ACM International Conference on Supercomputing*. 331–340.

[76] Samuel Greengard. 2020. Will RISC-V revolutionize computing? *Communications of the ACM* 63, 5 (2020), 30–32.

[77] Matthias Gries. 2004. Methods for evaluating and covering the design space during early design development. *Integration* 38, 2 (2004), 131–183.

[78] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. 2007. *System Design with SystemCTM*. Springer Science & Business Media.

[79] Ashok Halambi, Peter Grun, Vijay Ganesh, Asheesh Khare, Nikil Dutt, and Alex Nicolau. 2008. EXPRESSION: A language for architecture exploration through compiler/simulator retargetability. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'08)*. 31–45.

[80] Sodam Han, Yonghee Yun, Young Hwan Kim, and Seokhyeong Kang. 2020. Proactive scenario characteristic-aware online power management on mobile systems. *IEEE Access* 8 (2020), 69695–69711.

[81] Vinay Hanumaiah, Digant Desai, Benjamin Gaudette, Carole-Jean Wu, and Sarma Vrudhula. 2014. STEAM: A smart temperature and energy aware multicore controller. *ACM Transactions on Embedded Computing Systems* 13, 5s (2014), 1–25.

[82] Vinay Hanumaiah and Sarma Vrudhula. 2012. Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling. *IEEE Transactions on Computers* 63, 2 (2012), 349–360.

[83] ODROID Wiki. [n.d.]. Hardkernel. ODROID-XU3. Retrieved May 15, 2022 from https://wiki.odroid.com/old_product/odroid-xu3/odroid-xu3.

[84] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[85] John Hennessy and David Patterson. 2018. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*.

[86] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. *Communications of the ACM* 62, 2 (2019), 48–60.

[87] Jingcao Hu and Radu Marculescu. 2005. Energy-and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24, 4 (2005), 551–562.

[88] Zhengbing Hu, Qingying Zhang, Sergey Petoukhov, and Matthew He. 2021. *Advances in Artificial Systems for Logistics Engineering*. Springer.

[89] Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. 2011. Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *Proceedings of the 7th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. 247–256.

[90] Lin Huang, Feng Yuan, and Qiang Xu. 2009. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'09)*. 51–56.

[91] Michael Hutter and Jörn-Marc Schmidt. 2013. The temperature side channel and heating fault attacks. In *Proceedings of the International Conference on Smart Card Research and Advanced Applications*. 219–235.

[92] Peter Jamieson, Kenneth B. Kent, Farnaz Gharibian, and Lesley Shannon. 2010. Odin II—An open-source Verilog HDL synthesis tool for CAD research. In *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. 149–156.

[93] James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

[94] Shiyuan Jin, Guy Schiavone, and Damla Turgut. 2008. A performance study of multiprocessor task scheduling algorithms. *Journal of Supercomputing* 43, 1 (2008), 77–97.

[95] Norman P. Jouppi and David W. Wall. 1989. Available instruction-level parallelism for superscalar and superpipelined machines. *ACM SIGARCH Computer Architecture News* 17, 2 (1989), 272–282.

[96] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, et al. 2021. Ten lessons from three generations shaped Google's TPUv4i: Industrial product. In *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA'21)*. 1–14.

[97] Mostafa Khamis, Sameh El-Ashry, Ahmed Shalaby, Mohamed AbdElsalam, and M. Watheq El-Kharashi. 2018. A configurable RISC-V for NoC-based MPSoCs: A framework for hardware emulation. In *Proceedings of the 11th International Workshop on Network on Chip Architectures (NoCArc'18)*. 1–6.

[98] Sung Kim, Morteza Fayazi, Alhad Daftardar, Kuan-Yu Chen, Jielun Tan, Subhankar Pal, Tutu Ajayi, et al. 2022. Versa: A 36-core systolic multiprocessor with dynamically reconfigurable interconnect and memory. *IEEE Journal of Solid-State Circuits* 57, 4 (2022), 986–998.

[99] Joonho Kong, Sung Woo Chung, and Kevin Skadron. 2012. Recent thermal management techniques for microprocessors. *ACM Computing Surveys* 44, 3 (2012), 1–42.

[100] Maria Kotsifakou, Prakalp Srivastava, Matthew D. Sinclair, Rakesh Komuravelli, Vikram Adve, and Sarita Adve. 2018. HPVM: Heterogeneous parallel virtual machine. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 68–80.

[101] Anish Krishnakumar, Samet E. Arda, A. Alper Goksoy, Sumit K. Mandal, Umit Y. Ogras, Anderson L. Sartor, and Radu Marculescu. 2020. Runtime task scheduling using imitation learning for heterogeneous many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 4064–4077.

[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* 25 (2012).

[103] Vipin Kumar Kukkala, Sudeep Pasricha, and Thomas Bradley. 2020. SEDAN: Security-aware design of time-critical automotive networks. *IEEE Transactions on Vehicular Technology* 69, 8 (2020), 9017–9030.

[104] Andreas Kurth, Pirmin Vogel, Alessandro Capotondi, Andrea Marongiu, and Luca Benini. 2017. HERO: Heterogeneous embedded research platform for exploring RISC-V manycore accelerators on FPGA. *arXiv preprint arXiv:1712.06497* (2017).

[105] Henning Lategahn, Andreas Geiger, and Bernd Kitt. 2011. Visual SLAM for autonomous ground vehicles. In *Proceedings of the International Conference on Robotics and Automation.* 1732–1737.

[106] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization.* 75–86.

[107] Seyong Lee, Seung-Jai Min, and Rudolf Eigenmann. 2009. OpenMP to GPGPU: A compiler framework for automatic translation and optimization. *ACM SIGPLAN Notices* 44, 4 (2009), 101–110.

[108] Ching-Chi Lin, You-Cheng Syu, Chao-Jui Chang, Jan-Jan Wu, Pangfeng Liu, Po-Wen Cheng, and Wei-Te Hsu. 2015. Energy-efficient task scheduling for multi-core platforms with per-core DVFS. *Journal of Parallel and Distributed Computing* 86 (2015), 71–81.

[109] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, E. Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems.* 751–766.

[110] Chen Liu, Jeyavijayan Rajendran, Chengmo Yang, and Ramesh Karri. 2014. Shielding heterogeneous MPSoCs from untrustworthy 3PIPs through security-driven task scheduling. *IEEE Transactions on Emerging Topics in Computing* 2, 4 (2014), 461–472.

[111] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys* 52, 6 (2019), 1–39.

[112] Joshua Mack, Samet Arda, Umit Y. Ogras, and Ali Akoglu. 2021. Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices. *IEEE Transactions on Parallel and Distributed Systems* 33 (2021), 2148–2162.

[113] Joshua Mack, Sahil Hassan, Nirmal Kumbhare, Miguel Castro Gonzalez, and Ali Akoglu. 2022. CEDR—A compiler-integrated, extensible DSSoC runtime. *ACM Transactions on Embedded Computing Systems.* Online, April 13, 2022.

[114] Joshua Mack, Nirmal Kumbhare, Anish Krishnakumar, Umit Y. Ogras, and Ali Akoglu. 2020. User-space emulation framework for domain-specific SoC design. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'20).* 44–53.

[115] Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. 2021. SAGE: A split-architecture methodology for efficient end-to-end autonomous vehicle control. *ACM Transactions on Embedded Computing Systems* 20, 5s (2021), 1–22.

[116] Dipan Kumar Mandal, Srivatsava Jandhyala, Om J. Omer, Gurpreet S. Kalsi, Biji George, Gopi Neela, Santhosh Kumar Rethinagiri, et al. 2019. Visual inertial odometry at the edge: A hardware-software co-design approach for ultra-low latency and power. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'19).* 960–963.

[117] Sumit K. Mandal, Ganapati Bhat, Chetan Arvind Patil, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. 2019. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

[118] Sumit K. Mandal, Anish Krishnakumar, and Umit Y. Ogras. 2021. Energy-efficient networks-on-chip architectures: Design and run-time optimization. In *Network-on-Chip Security and Privacy.* Springer, 55–75.

[119] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. 2020. Agile SoC development with open ESP. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'20)*. 1–9.

[120] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the ACM Workshop on Hot Topics in Networks*. 50–56.

[121] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19)*. ACM, New York, NY, 270–288.

[122] Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. 2008. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 1 (2008), 3–21.

[123] Laurent Marsan and Marie-France Sagot. 2000. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of Computational Biology* 7, 3-4 (2000), 345–362.

[124] John R. Mashey. 2021. Interactions, impacts, and coincidences of the first golden age of computer architecture. *IEEE Micro* 41, 6 (2021), 131–139.

[125] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Computing Surveys* 37, 4 (2005), 316–344.

[126] Sparsh Mittal. 2020. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications* 32, 4 (2020), 1109–1139.

[127] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys* 47, 4 (2015), 1–35.

[128] Kasra Moazzemi, Biswadip Maity, Saehanseul Yi, Amir M. Rahmani, and Nikil Dutt. 2019. HESSLE-FREE: Heterogeneous systems leveraging fuzzy control for runtime resource management. *ACM Transactions on Embedded Computing Systems* 18, 5s (2019), 1–19.

[129] Ashwin Vishnu Mohanan, Cyrille Bonamy, and Pierre Augier. 2018. FluidFFT: Common API (C++ and Python) for fast Fourier transform HPC libraries. *arXiv preprint arXiv:1807.01775* (2018).

[130] Fabrizio Mulas, David Atienza, Andrea Acquaviva, Salvatore Carta, Luca Benini, and Giovanni De Micheli. 2009. Thermal balancing policy for multiprocessor stream computing platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 12 (2009), 1870–1882.

[131] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*. 1466–1482.

[132] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered insecure: GPU side channel attacks are practical. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*. 2139–2153.

[133] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman Jouppi, and David Patterson. 2021. The design process for Google's training chips: TPUv2 and TPUv3. *IEEE Micro* 41, 2 (2021), 56–63.

[134] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. 2019. Deep learning vs. traditional computer vision. In *Proceedings of the Science and Information Conference*. 128–144.

[135] Edson Luiz Padoin, Laércio Lima Pilla, Márcio Castro, Francieli Z. Boito, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut. 2015. Performance/energy trade-off in scientific computing: The case of ARM big.LITTLE and Intel Sandy Bridge. *IET Computers & Digital Techniques* 9, 1 (2015), 27–35.

[136] Zhixin Pan and Prabhat Mishra. 2021. Automated test generation for hardware Trojan detection using reinforcement learning. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 408–413.

[137] Sudeep Pasricha, Raid Ayoub, Michael Kishinevsky, Sumit K. Mandal, and Umit Y. Ogras. 2020. A survey on energy management for mobile and IoT devices. *IEEE Design & Test* 37, 5 (2020), 7–24.

[138] David Patterson. 2018. 50 years of computer architecture: From the mainframe CPU to the domain-specific TPU and the open RISC-V instruction set. In *Proceedings of the 2018 IEEE International Solid-State Circuits Conference-(ISSCC'18)*. IEEE, Los Alamitos, CA, 27–31.

[139] Arturo Pérez, Alfonso Rodríguez, Andrés Otero, David González Arjona, Alvaro Jiménez-Peralo, Miguel Ángel Verdugo, and Eduardo De La Torre. 2020. Run-time reconfigurable MPSoC-based on-board processor for vision-based space navigation. *IEEE Access* 8 (2020), 59891–59905.

[140] Martín Pi Puig, Laura Cristina De Giusti, Marcelo Naiouf, and Armando Eduardo De Giusti. 2019. A study of hardware performance counters selection for cross architectural GPU power modeling. In *XXV Congreso Argentino de Ciencias de la Computación (CACIC'19)*.

[141] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. 2006. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers* 55, 2 (2006), 99–112.

[142] Ivens Portugal, Paulo Alencar, and Donald Cowan. 2018. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications* 97 (2018), 205–227.

[143] Jing Pu, Steven Bell, Xuan Yang, Jeff Setter, Stephen Richardson, Jonathan Ragan-Kelley, and Mark Horowitz. 2017. Programming heterogeneous systems from an image processing DSL. *ACM Transactions on Architecture and Code Optimization* 14, 3 (2017), 1–25.

[144] Timo Punkka. 2012. Agile hardware and co-design. In *Proceedings of the Embedded Systems Conference*. 1–8.

[145] Jonathan Ragan-Kelley, Andrew Adams, Dillon Sharlet, Connelly Barnes, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2017. Halide: Decoupling algorithms from schedules for high-performance image processing. *Communications of the ACM* 61, 1 (2017), 106–115.

[146] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. Inter-cluster thread-to-core mapping and DVFS on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems* 4, 3 (2017), 369–382.

[147] Teresa Riesgo, Yago Torroja, and Eduardo De la Torre. 1999. Design methodologies based on hardware description languages. *IEEE Transactions on Industrial Electronics* 46, 1 (1999), 3–12.

[148] Tajana Simunic Rosing, Kresimir Mihic, and Giovanni De Micheli. 2007. Power and reliability management of SoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15, 4 (2007), 391–403.

[149] Ahmed Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy. 2009. Efficient FPGA implementation of FFT/IFFT processor. *International Journal of Circuits, Systems and Signal Processing* 3, 3 (2009), 103–110.

[150] Onur Sahin and Ayse K. Coskun. 2016. Providing sustainable performance in thermally constrained mobile devices. In *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*. 72–77.

[151] Santanu Sarma and Nikil Dutt. 2014. FPGA emulation and prototyping of a cyberphysical-system-on-chip (CPSoC). In *Proceedings of the IEEE International Symposium on Rapid System Prototyping*. 121–127.

[152] Anderson L. Sartor, Anish Krishnakumar, Samet E. Arda, Umit Y. Ogras, and Radu Marculescu. 2020. HiLITE: Hierarchical and lightweight imitation learning for power management of embedded SoCs. *IEEE Computer Architecture Letters* 19, 1 (2020), 63–67.

[153] Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. 2016. Co-designing accelerators and SoC interfaces using gem5-Aladdin. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12.

[154] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[155] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013. Mapping on multi/many-core systems: Survey of current and emerging trends. In *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13)*. 1–10.

[156] David B. Skillicorn and Domenico Talia. 1998. Models and languages for parallel computation. *ACM Computing Surveys*, 2 (1998), 123–169.

[157] Kyle L. Spafford and Jeffrey S. Vetter. 2012. Aspen: A domain specific language for performance modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage, and Analysis (SC'12)*. 1–11.

[158] Ashley Stevens. 2014. *Quality of Service (QoS) in ARM® Systems: An Overview*. White Paper. ARM, Cambridge, UK.

[159] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-Sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 16–25.

[160] Arvind K. Sujeeth, Kevin J. Brown, Hyoukjoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2014. Delite: A compiler architecture for performance-oriented embedded domain-specific languages. *ACM Transactions on Embedded Computing Systems* 13, 4s (2014), 1–25.

[161] Leonardo Suriano, Daniel Madroñal, Alfonso Rodríguez, Eduardo Juárez, César Sanz, and Eduardo de la Torre. 2018. A unified hardware/software monitoring method for reconfigurable computing architectures using PAPI. In *Proceedings of the 13th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC'18)*. 1–8.

[162] Karthik Swaminathan and Augusto Vega. 2021. Hardware specialization: From cell to heterogeneous microprocessors everywhere. *IEEE Micro* 41, 6 (2021), 112–120.

[163] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[164] Zhuo Tang, Ling Qi, Zhenzhen Cheng, Kenli Li, Samee U. Khan, and Keqin Li. 2016. An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment. *Journal of Grid Computing* 14, 1 (2016), 55–74.

[165] Umair Ullah Tariq, Hui Wu, and Suhaimi Abd Ishak. 2018. Energy-aware scheduling of conditional task graphs on NoC-based MPSoCs. In *Proceedings of the 51st Hawaii International Conference on System Sciences*.

[166] Thomas N. Theis and H.-S. Philip Wong. 2017. The end of Moore's law: A new beginning for information technology. *Computing in Science & Engineering* 19, 2 (2017), 41–50.

[167] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274.

[168] Yvan Tortorella, Luca Bertaccini, Davide Rossi, Luca Benini, and Francesco Conti. 2022. RedMulE: A compact FP16 matrix-multiplication accelerator for adaptive deep learning on RISC-V-based ultra-low-power SoCs. *arXiv preprint arXiv:2204.11192* (2022).

[169] Fengbin Tu, Shouyi Yin, Peng Ouyang, Shibin Tang, Leibo Liu, and Shaojun Wei. 2017. Deep convolutional neural network architecture with reconfigurable computation patterns. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 8 (2017), 2220–2233.

[170] Richard Uhrie, Daniel W. Bliss, Chaitali Chakrabarti, Umit Y. Ogras, and John Brunhaver. 2019. Machine understanding of domain computation for domain-specific system-on-chips (DSSoC). In *Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2019*, Vol. 11015. International Society for Optics and Photonics, SPIE, 180–187.

[171] Richard Uhrie, Chaitali Chakrabarti, and John Brunhaver. 2020. Automated parallel kernel extraction from dynamic application traces. *arXiv preprint arXiv:2001.09995* (2020).

[172] J. D. Ullman. 1975. NP-complete scheduling problems. *Journal of Computer and System Sciences* 10, 3 (1975), 384–393. https://doi.org/10.1016/S0022-0000(75)80008-0

[173] Peter Van Stralen and Andy Pimentel. 2010. Scenario-based design space exploration of MPSoCs. In *Proceedings of the IEEE International Conference on Computer Design*. 305–312.

[174] Prashant Varanasi and Gernot Heiser. 2011. Hardware-supported virtualization on ARM. In *Proceedings of the 2nd Asia-Pacific Workshop on Systems*. 1–5.

[175] Augusto Vega, John-David Wellman, Hubertus Franke, Alper Buyuktosunoglu, Pradip Bose, Aporva Amarnath, Hiwot Kassa, Subhankar Pal, and Ronald Dreslinski. 2021. STOMP: Agile evaluation of scheduling policies in heterogeneous multi-processors. In *Proceedings of the 3rd International Workshop on Domain Specific System Architecture in Conjunction with the 27th IEEE International Symposium on High-Performance Computer Architecture (DOSSA-3 @ HPCA'21)*.

[176] Nicolas Ventroux, Alexandre Guerre, Tanguy Sassolas, L. Moutaoukil, Guillaume Blanc, Charly Bechara, and Raphaël David. 2010. SESAM: An MPSoC simulation environment for dynamic application processing. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology*. 1880–1886.

[177] Matthew J. P. Walker and Jason H. Anderson. 2019. Generic connectivity-based CGRA mapping via integer linear programming. In *Proceedings of the Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'19)*. 65–73.

[178] Bo Wang, Sheng Ma, Guoyi Zhu, Xiao Yi, and Rui Xu. 2022. A novel systolic array processor with dynamic dataflows. *Integration* 85 (2022), 42–47.

[179] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi*^{TM}. Springer, 167–188.

[180] Liang Wang and Kevin Skadron. 2013. Implications of the power wall: Dim cores and reconfigurable logic. *IEEE Micro* 33, 5 (2013), 40–48.

[181] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[182] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the 54th Annual Design Automation Conference*. 1–6.

[183] Jenna Wiens and Erica S. Shenoy. 2018. Machine learning for healthcare: On the verge of a major shift in healthcare epidemiology. *Clinical Infectious Diseases* 66, 1 (2018), 149–153.

[184] Dhananjaya Wijerathne, Zhaoying Li, Anuj Pathania, Tulika Mitra, and Lothar Thiele. 2021. HiMap: Fast and scalable high-quality mapping on CGRA via hierarchical abstraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 10 (2021), 3290–3303.

[185] Yen-Kuan Wu, Shervin Sharifi, and Tajana Simunic Rosing. 2011. Distributed thermal management for embedded heterogeneous MPSoCs with dedicated hardware accelerators. In *Proceedings of the IEEE 29th International Conference on Computer Design (ICCD'11)*. 183–189.

[186] Yi Xiang and Sudeep Pasricha. 2015. Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting. *IEEE Transactions on Multi-Scale Computing Systems* 1, 4 (2015), 220–235.

[187] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2019. Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 6 (2019), 1416–1427.

[188] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2021. Plasticity-on-chip design: Exploiting self-similarity for data communications. *IEEE Transactions on Computers* 70, 6 (2021), 950–962.

[189] Yan Xiong, Jian Zhou, Subhankar Pal, David Blaauw, Hun-Seok Kim, Trevor Mudge, Ronald Dreslinski, and Chaitali Chakrabarti. 2020. Accelerating deep neural network computation on a low power reconfigurable architecture. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'20)*. 1–5.

[190] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. 161–170.

[191] Yunming Zhang, Mengjiao Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, and Saman Amarasinghe. 2018. Graphlt: A high-performance graph DSL. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), Article 121, 30 pages.

[192] Zhongyuan Zhao, Weiguang Sheng, Qin Wang, Wenzhi Yin, Pengfei Ye, Jinchao Li, and Zhigang Mao. 2020. Towards higher performance and robust compilation for CGRA modulo scheduling. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2201–2219.

[193] Junlong Zhou, Jin Sun, Peijin Cong, Zhe Liu, Xiumin Zhou, Tongquan Wei, and Shiyan Hu. 2019. Security-critical energy-aware task scheduling for heterogeneous real-time MPSoCs in IoT. *IEEE Transactions on Services Computing* 13, 4 (2019), 745–758.

[194] Junlong Zhou, Mingyue Zhang, Jin Sun, Tian Wang, Xiumin Zhou, and Shiyan Hu. 2022. DRHEFT: Deadline-constrained reliability-aware HEFT algorithm for real-time heterogeneous MPSoC systems. *IEEE Transactions on Reliability* 71, 1 (2022), 178–189.

Received 5 July 2022; revised 13 July 2022; accepted 10 August 2022